Proceedings of SPIE, Vol. 3693. Presented at *13th Annual AeroSense*, Orlando, FL, April 1999.

Vision based obstacle detection and path planning for planetary rovers

Magnús Snorrason[a], Jeff Norris[b], and Paul Backes[c]

[a,b]Charles River Analytics, 725 Concord Ave., Cambridge, MA 02138
[c]Jet Propulsion Laboratory, Pasadena, CA 91109

ABSTRACT

NASA's next generation Mars rovers are capable of capturing panoramic stereo imagery of their surroundings. Three-dimensional terrain maps can be derived from such imagery through stereo image correlation. While 3-D data is inherently valuable for planning a path through local terrain, obstacle detection is not fully reliable due to anomalies and noise in the range data. We present an obstacle-detection approach that first identifies potential obstacles based on color-contrast in the monocular imagery and then uses the 3-D data to project all detected obstacles into a 2-D overhead-view obstacle map where noise originating from the 3-D data is easily removed. We also developed a specialized version of the A* search algorithm that produces optimally efficient paths through the obstacle map. These paths are of similar quality as those generated by the traditional A*, at a fraction of the computational cost. Performance gains by an order of magnitude are accomplished by a two-stage approach that leverages the specificity of obstacle shape on Mars. The first stage uses depth-first A* to quickly generate a somewhat sub-optimal path through the obstacle map. The following refinement stage efficiently eliminates all extraneous way-points. Our implementation of these algorithms is being integrated into NASA's award-winning Web-Interface for Telescience.

Keywords: Rover navigation, obstacle detection, path planning

1. INTRODUCTION

The Jet Propulsion Laboratory's Web Interface for TeleScience (WITS, http://telerobotics.jpl.nasa.gov/tasks/wits/) is an important step towards involving scientists at their home institutions in the planning of daily mission activities of future Mars rovers. Mission scientists must choose daily mission goals and the rover's next destination. WITS facilitates this by providing a collaborative environment for visualization of imagery transmitted from Mars. However, mission scientists are not experts in rover navigation and they do not have time to consult with NASA's navigation experts for every potential destination.

We designed and prototyped a software tool that allows scientists to rapidly evaluate for themselves navigability of many different destinations, *before* submitting one for final verification by NASA's engineering team. We demonstrated feasibility by implementing an end-to-end obstacle detection & path planning system, verifying it on imagery from JPL's Mars Yard, and demonstrating how it can act as a server to WITS.

1.1 Summary of Objectives

The overall path-planning objective was broken down into the following five specific objectives, or tasks:

1. Obtain and analyze a panoramic set of images and associated 3-D data (derived from stereo) of a scenario similar to a Mars landing site
2. Develop an algorithm to analyze each image with the intent of locating obstacles that would be hazardous to a small rover (the size of a Sojourner/Rocky-7 Mars rover)
3. Develop an algorithm that combines the results across images to generate a unified "obstacle map" describing the location of each obstacle in the panorama
4. Develop an algorithm to rapidly suggest an optimal (or near-optimal) path through the obstacle map, i.e. the shortest obstacle-free path
5. Based on those algorithms, design and implement a software server that communicates with WITS, generating optimal paths to user-specified destinations fast enough for convenient user interaction

[a] Email: mss@cra.com; Telephone: +1-617-491-3474x524; http://cns-web.bu.edu/~snorraso
[b] Email: jsnorris@mit.edu
[c] Email: paul.g.backes@jpl.nasa.gov

1.2 Summary of Approaches

The five taks listed above were approached as follows.

For the 1st task, we obtained a data set from JPL showing a panorama (consisting of 24 overlapping gray-scale images) of sand and rocks in the JPL Mars Yard. Analysis of this data set drove many of our design decisions and this data was used both for tuning and evaluating algorithms.

After analysis of this data, the decision was made to use gray-scale contrast as the basis for the 2nd task: discriminating obstacles (rocks) from ground (sand). Although the data had 3-D coordinate maps (elevation maps) associated with each image, we found these maps to be insufficient for reliable detection of obstacles. Conversely, we found that an automatic gray-scale-to-binary thresholding algorithm was able to correctly discriminate obstacle vs. ground for most pixels in each image. By biasing the thresholding algorithm and then cleaning up the resulting binary image with morphological image operations, we were able to construct obstacle masks that correctly labeled all pixels belonging to obstacles without adding false alarm obstacles.

Our approach to the 3rd task—combining the masks into one obstacle map for the whole panorama—was to project each mask into a top-down view and then stitch them together. This was possible because the elevation maps can be treated as look-up tables, associating each pixel in each obstacle mask with a (x,y) coordinate in a common frame of reference. The obstacle mask and obstacle map algorithms were demonstrated to work equally well for all 24 Mars Yard images using the same set of parameters values, but we expect these values may have to be tuned for other image sets.

Given this obstacle map, the 4th task was to develop an algorithm to rapidly generate a near-optimal path between the rover's current location and a user-specified destination. We defined "optimal" as the shortest obstacle-free path between these two locations, with the secondary requirement that the number of turns be minimized (shortest paths tend to hug obstacles, requiring many way-points). A near-optimal path might be slightly longer or have slightly more way-points, but it must still be obstacle-free. The A* search algorithm [1] is a popular approach to generating optimally short paths, but it does not address the number of way-points and we found it to be unacceptably slow for this application. We developed a "depth-first" version of the A* algorithm that finds a near-optimal path orders-of-magnitude faster than the regular A*. We then developed an algorithm to quickly refines that path by eliminating all unnecessary way-points. The refined path has no more than a few way-points per obstacle and is generally shorter (closer to optimal) than the depth-first-A* path.

Finally, we implemented a remotely accessible real-time path planning server that communicates with WITS. This server is implemented in Java—as is all other software in this project—and has been tested on both Unix and Windows hardware. WITS users can graphically specify destination points and then WITS sends these coordinates to our server. In a few seconds the server responds with a list of way-points and WITS draws the path in the panoramic or overhead view imagery. Our path planner can also be run independently of WITS, using a separate GUI.

2. PREVIOUS WORK IN ROVER PATH PLANNING

Autonomous mobile robot research has been underway for around 30 years. One of the first such robots, Shakey, was developed between 1966 and 1972 [2]. Already then was it recognized that visual perception was one of the most important issues. Other well known robotic ground vehicles capable of autonomous navigation include the HERMIES series of robots developed at the Oak Ridge National Laboratory [3] and the NAVLAB series of Autonomous Land Vehicles at Carnegie Mellon University [4]; [1]. Work on the Nomad rover at Carnegie Mellon and NASA Ames has touched on the need for path planning assistance in a telerobotic user interface, but no automation of obstacle detection or path planning is involved [5]. Similarly for the Russian Marsokhod rover (also in collaboration with NASA Ames), it uses a vision-based operator interface but no assistance is offered in path planning [6]. A number of very interesting projects have recently been conducted in Europe, such as the work of Oriolo, Ulivi & Vendittelli [7], which also uses a modified A* search algorithm, but in a very different environment (indoor hallways), and the work of Kolesnik, Paar, Bauer & Ulm [8], who produce "risk maps" online and automated path plans using Dijkstra's algorithm, but only for the region immediately local to their (conceptual) Lunar rover. At NASA JPL, under the Rocky 7 Mars rover project, work on obstacle detection has concentrated on terrain measurements based on stereo-derived elevation maps (rather than color-contrast as described here) and path planning has been concerned with avoiding the immediate next obstacle [9]; [10], rather than being goal-oriented like our project.

3. SYSTEM ARCHITECTURE

The overall architecture was split into two parts, based on a clear delineation of tasks. The first task is *obstacle map generation* from imagery that will be transmitted to Earth on a daily basis. The time taken to generate a map is not critical, so

long as it is on the order of tens of minutes rather than hours. Automation, however, is highly desirable, since that frees scientists to attend to other tasks. The second task is *path planning* to user-specified destinations. Unlike obstacle map generation, time is critical in this task because scientists want to evaluate many different paths—interactively—before committing to one.

3.1 Obstacle Map Generation

The following block diagram outlines our design for obstacle map generation.

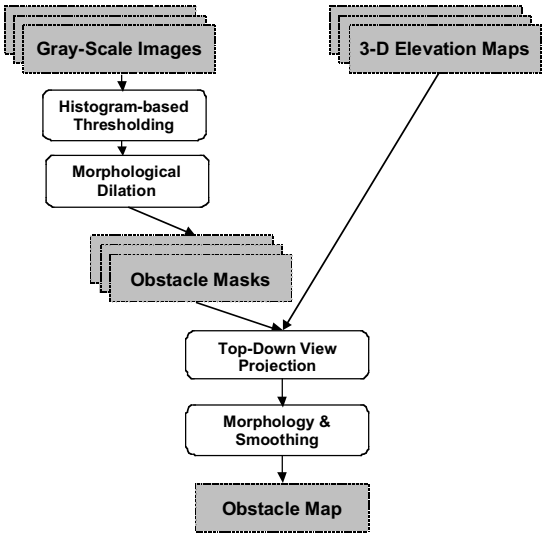

Figure 1: Obstacle Map Generation

The inputs that go into making an obstacle map for a region of interest consist of the full set of gray-scale images and associated elevation maps that together tile the region. In case of the JPL Mars Yard data, the gray-scale (one byte per pixel) images were a set of 24 JPEG images that were all captured with the same camera (the rover's left "eye") from the same location in the Mars Yard. The camera was pointed in a slightly different direction for each image, such that most of the Mars Yard was covered in two concentric circles, 12 images per circle. Figure 2 below shows an example of one such image, with the 3-D elevation map visualized in the form of a semi-transparent overlay.

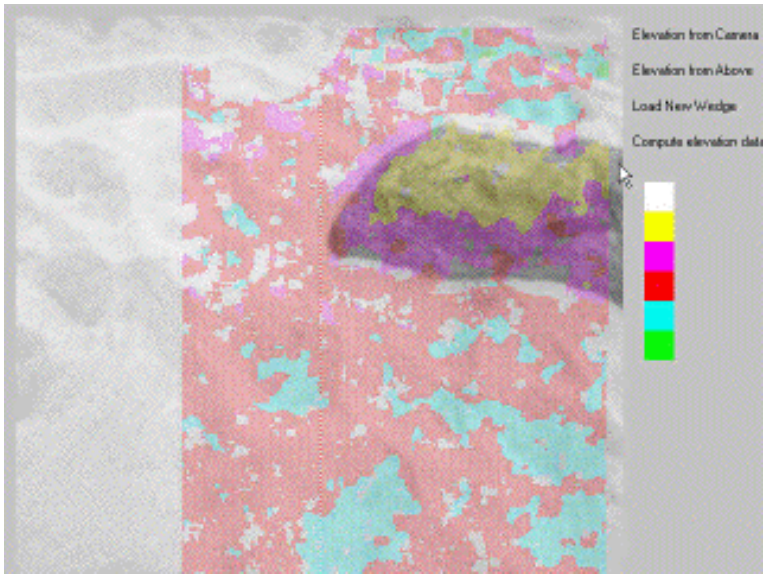

Figure 2: Input image with elevation data overlay

The 3-D elevation maps derive from the stereo disparity between this set of images and another set that was captured at the same time by a similar camera located adjacent to the first one on the same camera mount (the rover's right "eye"). Since

stereo disparity provides range information and the pointing vector for the two cameras is known for each image, it is possible (in theory) to know the 3-D location of each pixel in a camera-centered spherical coordinate system. The final step is then to transform to Cartesian coordinates that are referenced to the ground-plane, hence the resulting 3-D data is referred to as an elevation map. In practice however, due to the nature of stereo correlation, range information will be missing for many pixels in each image. Some regions are missing due to occlusion: when either camera cannot see behind a rock, no disparity can be computed for the occluded region. Other regions were also missing from the elevation maps we received from JPL, presumably because correlations could not be found between the left and right images. Finally, a fair number of scattered pixels had clearly erroneous location values, far outside the imaged region.

To summarize, elevation data may be intuitively well suited for detection of physical objects, but the elevation maps had a number of anomalies that would complicate obstacle detection. On the other hand, detecting rocks in the photographic images was a trivial task for human observers. In every image, the rocks were significantly darker than the surrounding sand. Rocks on Mars seem to also be visually distinct. Rocks seen in the Pathfinder imagery[1] generally have gray regions while the ground has only hues of brown.

As a result, we developed an algorithm that automatically generates a binary mask for each image, with rocks shown in white and everything else masked out as black. The two steps in generating such a mask—as indicated in the block diagram above—are a threshold-to-binary function followed by a morphological dilation operation. Figure 3 below shows an example of the results after each step.

Figure 3: Input image, after thresholding, and after morphological dilation

3.2 Thresholding

The middle panel in Figure 3 above shows the result of thresholding one image from the Mars Yard data set. It is close to being a usable obstacle mask as is, with no false alarm regions and the outline of the rock nearly perfect. The holes inside the rock region are the only problem, they get cleaned up using morphology as described in the next section.

The thresholding operation has two free parameters: threshold value and polarity. The former is automatically determined via iterative histogram analysis of the image [11]. Images such as these tend to have bimodal histograms, with one peak for the range of rock-colored gray-values and another peak for the sand-colored values. This algorithm tends to balance bimodal histograms evenly, placing the threshold midway between the peaks. That turns out to be generally correct, but some obstacles of similar color as the surrounding ground are missed using that threshold value. Hence, we bias the threshold slightly towards the higher valued (sand-colored) peak. The bias value was empirically chosen to minimize both false positives (which occur if the bias is too high) and missed detections (which occur if the bias is too low).

The free parameter of threshold value is thus replaced with a bias parameter to the histogram analysis algorithm. This is an improvement because it would be much harder (or impossible) to find a single threshold value that works for all images in a set than to find a single bias value that works overall.

The polarity was determined empirically for this data set. Rocks are always *darker* than the ground, hence values *below* the threshold are mapped to white in the mask. For a more general solution, an algorithm would be needed to automatically determine the polarity. For example, the assumption could be made that most images contain more pixels showing ground than obstacles, and hence the polarity could be determined such that which-ever side of the threshold tends to have fewer pixels is mapped to white (representing obstacles).

[1] http://mars.jpl.nasa.gov/MPF/index1.html

3.3 Morphological Dilation

Since rocks can have light-colored spots, small clusters of pixels can end up on the wrong side of the threshold. The mask produced by thresholding thus often has small holes in the obstacle region, as seen in the middle panel in Figure 3. As shown in the right-hand panel in the same figure, this is fixed by performing a morphological dilation operation [12]. Dilation has the secondary effect of slightly expanding the boundary of the obstacle. This is generally advantageous, since it removes high-frequency noise from the perimeter of obstacles.

The two free parameters of all morphological operations are the operator shape and size. We found the shape to be unimportant for this application, hence we chose a square operator because it allows for a very low computational cost implementation. The operator size remains a free parameter, but we did not find the quality of the result to be overly sensitive to that parameter's value.

3.4 Top-Down View Projection

At this point, we had one mask per input image and—as indicated by the block diagram at the beginning of the chapter—the next step was to combine the masks into one *obstacle map* for the whole panorama. Our approach was to perform an orthographic projection of each mask into a top-down view and then stitch them together. This is possible because the elevation maps can be treated as look-up tables, associating each pixel in each obstacle mask with a (*x,y*) coordinate in a common frame of reference (origin is on the ground, right below the camera; units are in meters).

The elevation map is pixel-by-pixel co-registered with the input image, so orthographic projection of the elevation data was done as follows. First, a top-down-view map was initialized to black (all zero) and then a white pixel was placed in this map at every (*x,y*) location listed in the elevation data. The left panel of Figure 4 below shows the top-down projection of the elevation data associated with the image shown in the figures above.

Since the elevation map is also co-registered with the obstacle mask for each image, the mask can thus be projected in a similar way. A top-down-view map is initialized to black and then one white pixel is placed in this top-down-view map for each white pixel in the mask, at the (*x,y*) location specified by the co-registered point in the elevation map. The right panel of Figure 4 below shows the top-down projection of the mask shown above in Figure 3. As the figure shows, the projected mask isolates a region of the elevation map (the one rock in this image), just the same way as the original mask isolated a region in the original image.

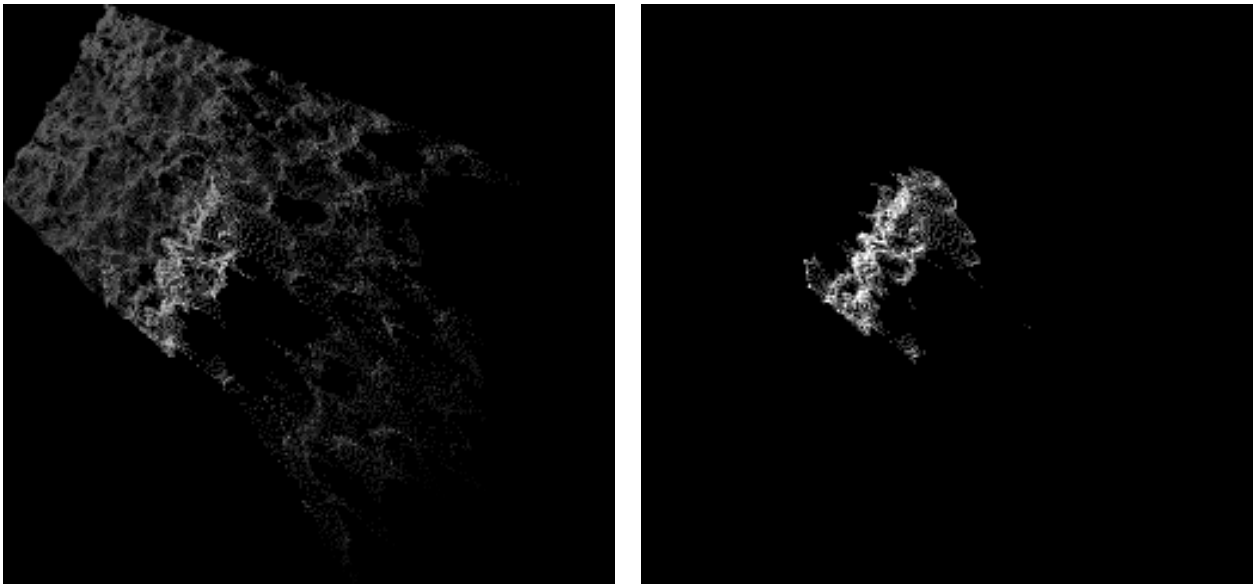

Figure 4: Elevation map (left) and obstacle mask (right) for one image projected to top-down view

Placing all the masks in a common coordinate system takes care of getting the individual masks to line up, but one task is still left to the user: determining the total size and scale of the top-down-view map. This is done by specifying the size of the final obstacle map in pixels and by specifying the size—in meters—of the region represented by the final map. Figure 5 below shows such a map that is 600 by 600 pixels and covers 10 by 10 meters, centered on the camera location. The boxed region indicates the mask shown above in Figure 4.

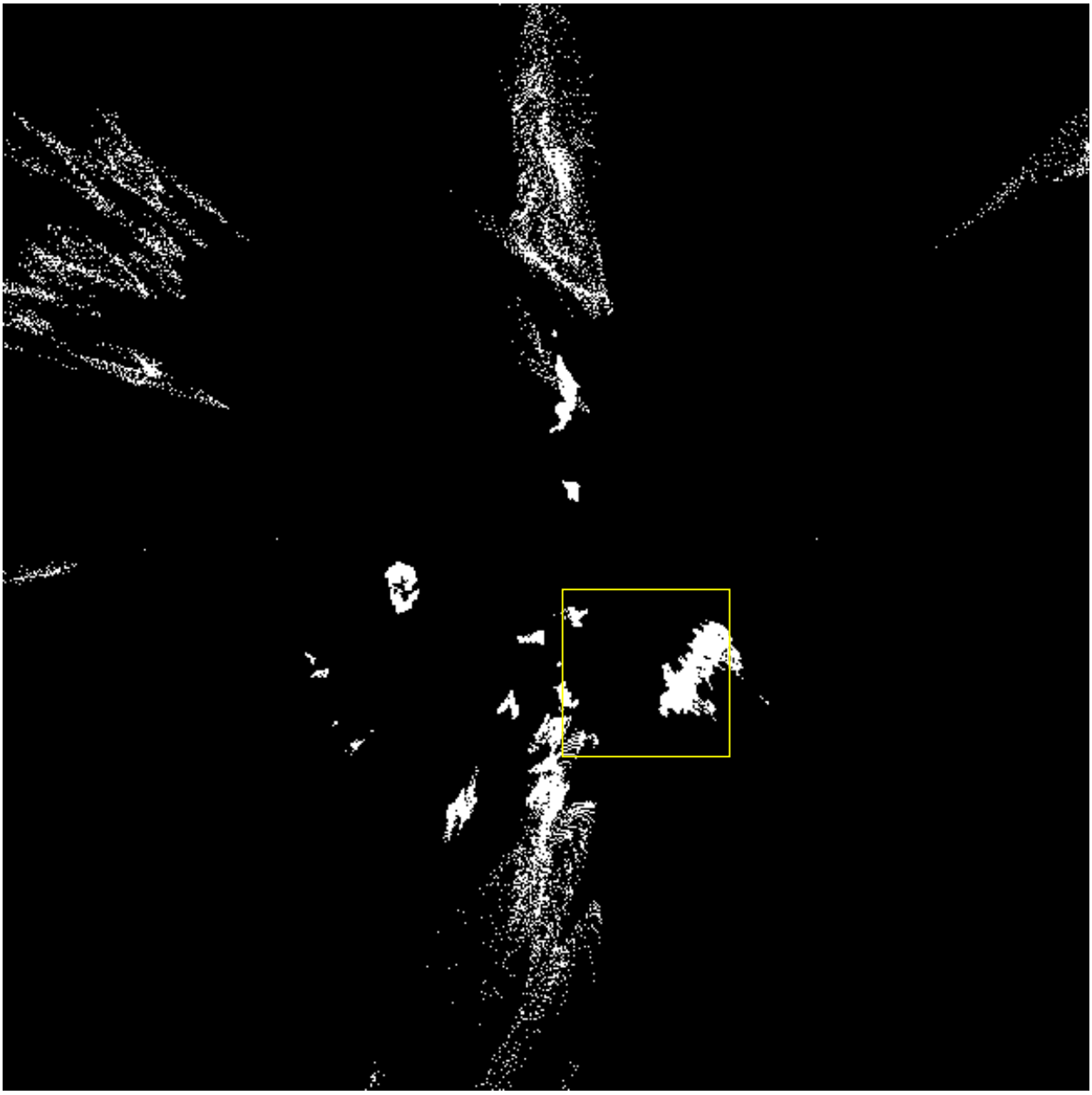

Figure 5: All masks combined into one top-down view

3.5 Morphology and Smoothing

It is clear from the figure above that this process introduces false alarm obstacles into the map. All the sparse white pixels (distributed near the edges and radiating out from the center of the map) are the results of incorrect (x,y) values in the elevation data. Due to these erroneous elevation data points, some white pixels in some masks get mapped to locations that are far from the correct locations that their neighbors get mapped to. This is why the final step in obstacle map generation is needed, to "clean up" the map.

Since most of those erroneous pixels are isolated and do not have white neighbors within a radius of a few pixels, they can be removed by morphological erosion [12] using a very small operator. Figure 6 below shows this effect, using a square operator of size 2 by 2. This is the smallest practical value for this operator and probably a safe choice for all maps in terms of not removing real obstacles (assuming a scale similar to this map or larger). Larger erosion operators will remove more false alarm obstacles, but may also remove valid obstacles.

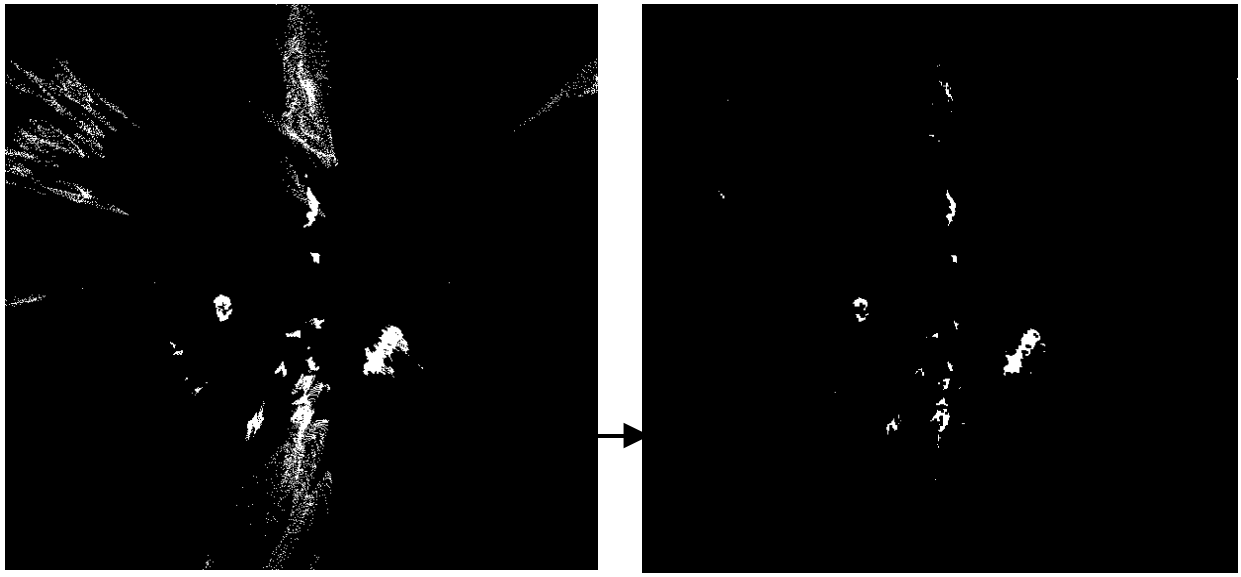

Figure 6: Obstacle map before and after erosion

A dilation must be performed to put back the boundary pixels that the erosion "shaved off" the real obstacles. Performing just that task would require a dilation operator of exactly the same size as the erosion operator. However, dilating with a larger operator takes care of the additional task of accounting for the width of the rover. In other words, by making each obstacle wider on all sides by at least half the width of the rover, path planning becomes simpler because paths only have to be one pixel wide. Figure 7 shows the effect of dilating by an operator of 8 by 8.

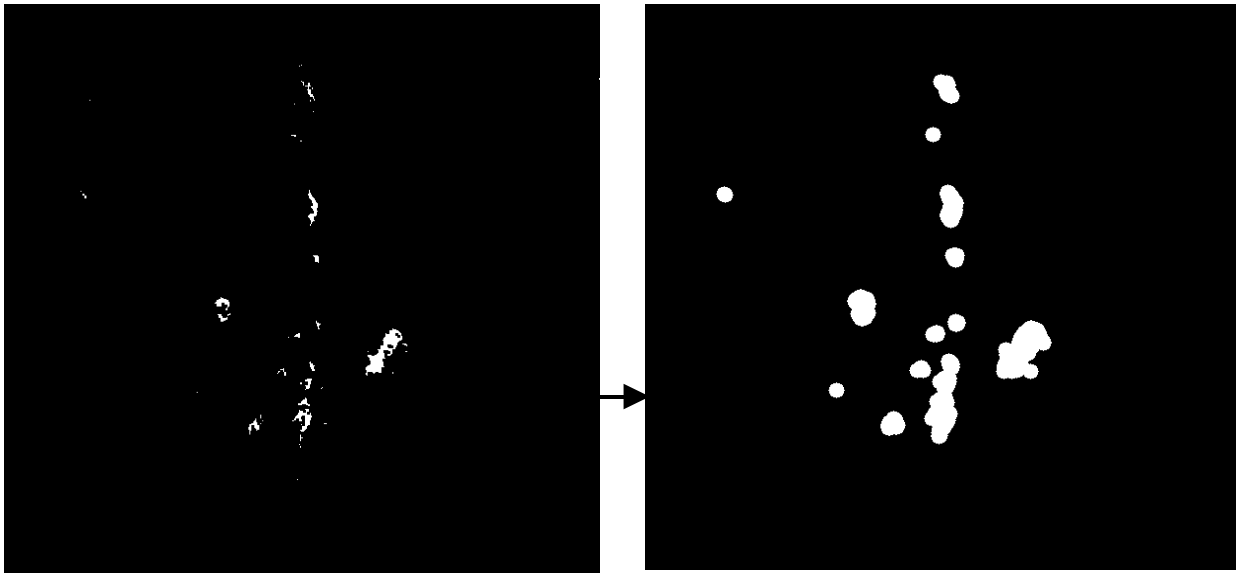

Figure 7: Obstacle map before and after dilation

Widening obstacles by more than half-a-rover's width introduces a safety margin. The path planner treats all white pixels the same—as impassible obstacles—and hence a ribbon of ground around the obstacle will be marked as impassible. However, another form of safety margin may also be useful, a *semi-passable* safety margin. This is accomplished by smoothing the dilated obstacle map, producing a gradient of gray values radiating out from every obstacle, as shown in Figure 8. By default, the path planner will only treat perfectly black (zero-valued) pixels as being completely safe, hence this gradient region is an additional safety margin. Unlike the solid white safety margin however, this gradient margin allows the path planner to treat the same obstacle map differently depending on a parameter telling the path planner how close to the obstacle it can go (i.e. the threshold value for a pixel to be considered passable).

Figure 8: Close-up of a final obstacle map showing smoothing

3.6 Path Planning

Given the completed obstacle map, our next task was to develop an algorithm to rapidly generate a near-optimal path between the rover's current location and a user-specified destination. As explained in the introduction, we defined "optimal" as the shortest obstacle-free path between these two locations, with a secondary requirement to minimize the number of turns.

With obstacle maps containing on the order of 10^5 or 10^6 pixels and possibly showing hundreds of obstacles, it was clear from the beginning that interactive generation of *optimal* paths might not be computationally feasible. One of the first algorithms we looked into was the A* search algorithm, a well know algorithm for the generation of optimally short paths [1]. We tested it on various paths in the map shown above (in Figure 7) and found it to be unacceptably slow. Running on a PC, it took from a few minutes to 15 minutes to generate optimally short paths in this 600x600 pixel map that contains 13 obstacles. That is already too slow for interactive use and would get much slower with larger maps and more obstacles. Finally, A* does not address the requirement of minimizing the number of turns. That would have to be taken care of with a node-reduction algorithm, replacing multi-segment curves with fewer and longer straight segments.

We realized that these issues were related: by using an algorithm that trades off optimality for speed, we could rapidly generate a path that might be quite sub-optimal before refinement, but after node-reduction might look very much like a similarly refined A* generated path.

3.7 Depth-First-A* Search

Hence, we developed a version of the A* algorithm that finds a less-than-optimal path at least two orders-of-magnitude faster than the regular A* algorithm. The two most important features of A* are as follows

- A* always explores the least-cost partial path available while deleting redundant paths
- A*'s cost function includes a strict lower bound estimate of the remaining distance to goal

We did not change the first feature since it contributes both to optimality and speed. The second feature forces A* to perform a more exhaustive search, consequently it contributes to optimality but at a great cost in speed.

Our solution was to compare with a heuristic *overestimate* of the remaining distance from any given point, rather than comparing with an underestimate of the remaining distance. The resulting search proceeds in a more "depth-first" manner than standard A*'s "breadth-first" approach. Also, our algorithm stops when it finds the *first* unobstructed path that reaches the goal, whereas A* will continue to search among paths that have potentially shorter last segments.

Figure 9 shows a hypothetical comparison of the two algorithms in a simple 6x6 search example where the dark square in the lower left corner is the starting position and the light square near the upper right corner is the destination. By its

breadth-first nature, A* (on the left) would follow all possible paths from the starting position for at least a few pixels before choosing the most direct one. Our depth-first approach (on the right) first tries the path that heads directly for the destination and upon finding it blocked, backtracks only as much as needed to get around it. Consequently, our approach finds a path that is slightly different from the most direct path, but it only has to consider about half as many pixels as standard A*.

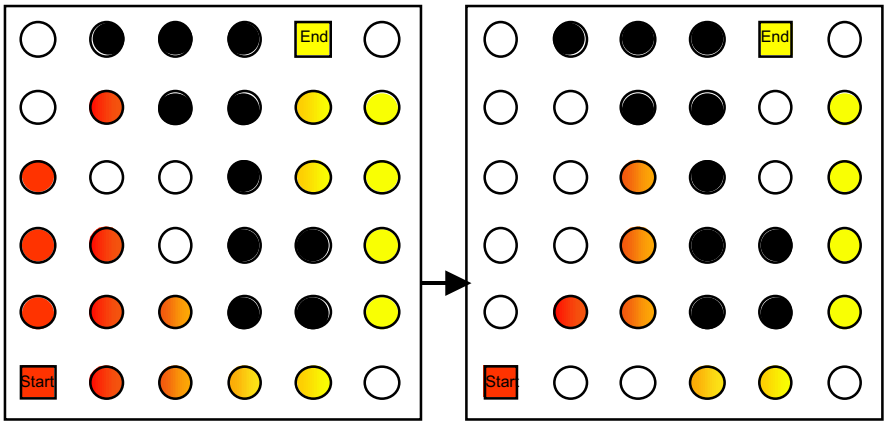

Figure 9: Pixels considered by standard A* (left) and depth-first-A* (right)

Figure 10 below demonstrates our search algorithm's approach to the same obstacle map as shown before in Figure 8. All pixels considered by the algorithm are colored in gray. Both issues described above are salient in this figure: not many pixels were considered, but the resulting path is not optimal (it dips down before each obstacle, rather than going straight above them). As for speed, this path took a few seconds to generate on a 166MHz Pentium laptop, more than fast enough for interactive usage.

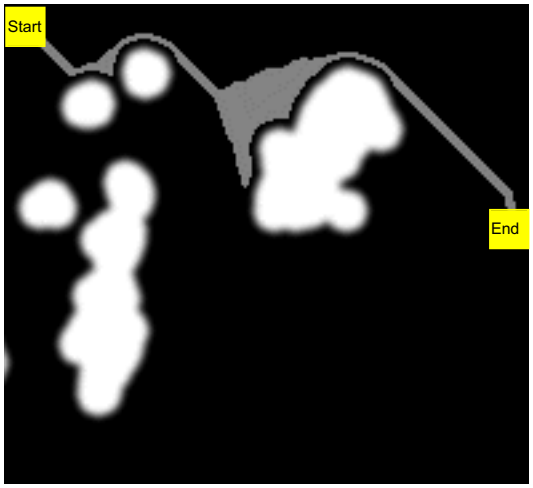

Figure 10: Close-up of obstacle map, marking pixels considered by depth-first-A*

3.8 Node-Reduction Path Refinement

The final step in our design process was to develop an algorithm to quickly refine the depth-first-A* path by eliminating all unnecessary way-points, or nodes. The result is an algorithm that starts at the destination and walks back along the path one pixel at the time until it can no longer see the destination. Then it goes forward again one pixel to the last pixel along the path where the destination was visible and draws a straight path along this "line-of-sight," eliminating any nodes it may have passed along the path so far. Next, the process starts all over again, with the current position as the destination. Figure 11 below shows how this process eliminates three nodes in the example from before.

Starting at the destination (the light colored square), our algorithm will follow the light colored circles node-by-node until it reaches the rightmost node in the bottom row. Then the destination is no longer in sight, so the algorithm back-tracks by one node and connects that node with a straight path to the destination, bypassing three nodes and eliminating one turn. On the next leg of the path the algorithm bypasses one node and eliminates one turn. On the third leg the algorithm has still

not lost sight of its current destination when it reaches the origin, hence this becomes the final leg of the path, bypassing two nodes and eliminating two turns. In this simple case, the final path is identical in length to the optimal path generated by traditional A*.

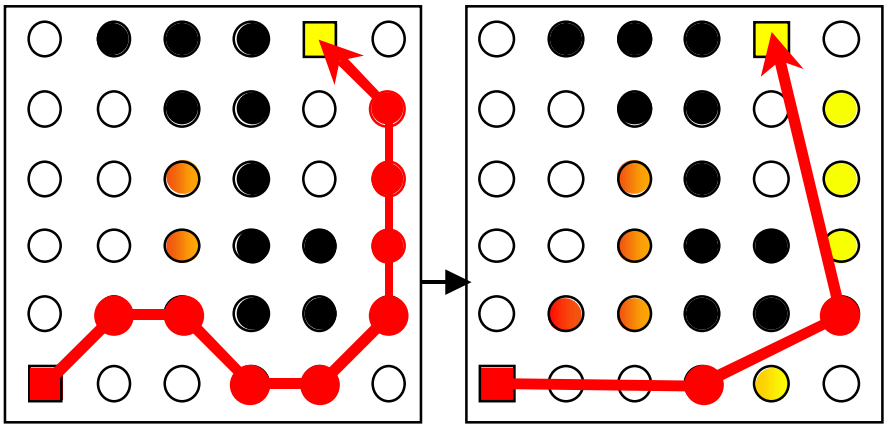

Figure 11: Depth-first-A* path before (left) and after path refinement (right)

As demonstrated by this simple example, removing nodes can be a low-computational-cost approach to turning sub-optimal paths into optimal paths. In particular, this path refinement approach is much simpler than any approach that tries to move nodes around. As described so far however, our method is unable to remove nodes where the path hugs the boundary of a convex obstacle. In such a situation, nodes are rapidly obscured by the obstacle as the algorithm follows the curve. Therefore, we added one more modification: having the node-removal algorithm use a slightly smaller safety margin than the depth-first-A* algorithm. This is done as follows:

- Boost safety margins on obstacles during the smoothing step of obstacle map generation
- For the depth-first-A* algorithm, set the maximum allowable gray-scale value = 0 for a pixel to be considered passable
- For the node-removal algorithm, set the same parameter to a value > 0

Effectively, this means that the node-removal algorithm is given permission to "shave corners" where the depth-first-A* algorithm was forced to take artificially wide turns. Figure 12 below demonstrates that this algorithm is very effective with real obstacle maps. Depth-first-A* generated a sub-optimal path (upper edge of the gray path) with a large number of nodes. The node-removal algorithm eliminated all but one of those nodes, leaving an optimal path. As for speed, there was no appreciable increase in total run-times when this node-removal algorithm was added to the path planning software.

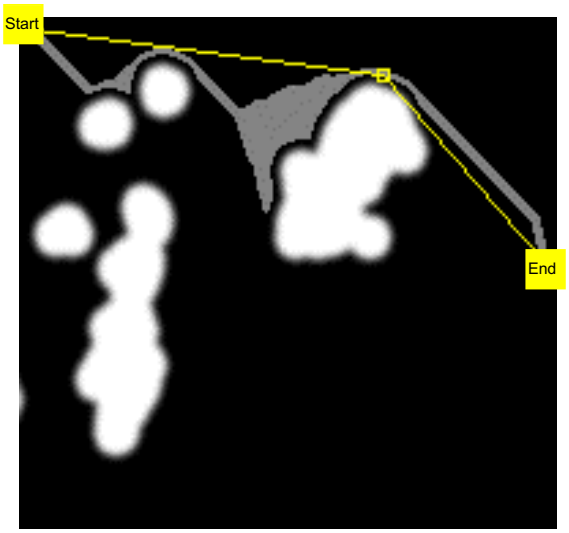

Figure 12: Depth-first-A* considered pixels (gray) and refined path (yellow)

It is interesting to note that this path planning approach derives great performance improvements from the fact that obstacles are typically convex and relatively compact. As a counter-example, the depth-first-A* algorithm would not be

efficient in a labyrinth-like environment of long winding corridors (i.e., non-compact obstacles). It would persist in following the first corridor that starts out in approximately the correct direction, investing significant effort in this potential path even when others are just as likely to reach the goal.

Since Mars terrain can be described as strewn with weathered rocks, such labyrinth-like situations are unlikely to occur. The likeliest situation is that the path must snake around obstacles, where each obstacle should add no more than a few way-points. This is the type of scenario our path planning algorithms are optimized for, and as demonstrated in the examples above, our approach succeeds in generating optimal or near-optimal paths in a small fraction of the time taken by traditional search algorithms.

4. CONCLUSIONS

JPL's Web Interface for TeleScience (WITS) is an important step towards involving principal investigators and other scientists outside NASA centers in the planning of daily mission activities. It enables scientists at their home institutions to analyze imagery of terrain within view of the rover, using the web. We demonstrated an extension of WITS that provides visualizations of optimal paths to user-selected destinations. WITS communicates with our software in real-time, which computes the paths online based on an obstacle map computed from down-linked sensor data. In particular, we conclude the following points:

- Autonomous rover navigation can be effectively split into obstacle map generation and path planning
- Color-based obstacle detection can be at least as reliable as range-based detection
- More data sets are needed to evaluate robustness of the color-based approach in different terrains
- Overhead maps provide a convenient and efficient terrain representation for path planning
- Depth-first A* search followed by iterative node-removal is extremely fast and generates optimal or near optimal paths

5. ACKNOWLEDGEMENTS

This work was supported by NASA contract No. NAS8-98192 with the Jet Propulsion Laboratory, Pasadena, California.

6. REFERENCES

[1] A. Stenz and M. Hebert, "A Complete Navigation System for Goal Acquisition in Unknown Environments," presented at ARPA Image Understanding Workshop, Monterey, CA, 1994.

[2] N. J. Nilsson, "Shakey the Robot," SRI AI Center, Technical Report 323, April, 1984 1984.

[3] B. L. Burks, G. de Saussure, C. R. Weisbin, J. P. Jones, and W. R. Hamel, "Autonomous Navigation, Exploration, and Recognition Using the HERMIES-IIB Robot," in *IEEE Expert*, vol. Winter, 1987, pp. 18-27.

[4] Y. Goto and A. Stenz, "Mobile Robot Navigation: The CMU System," in *IEEE Expert*, vol. Winter, 1987, pp. 44-54.

[5] D. Wettergreen, M. Bualat, D. Christian, K. Schwehr, H. Thomas, D. Tucker, and E. Zbinden, "Operating Nomad during the Atacama Desert Trek," presented at Field and Service Robotics Conference, Canberra, Australia, 1997.

[6] D. Wettergreen, H. Thomas, and M. Bualat, "Initial Results from Vision-based Control of the Ames Marsokhod Rover," presented at IEEE/RSJ International Conference on Intelligent Robots and Systems, Control of Wheeled Robots, Grenoble, France, 1997.

[7] G. Oriolo, G. Ulivi, and M. Vendittelli, "Real-Time Map Building and Navigation for Autonomous Robots in Unknown Environments," *IEEE Trans. Systems, Man, & Cybernetics-B*, vol. 28, pp. 316-333, 1998.

[8] M. Kolesnik, G. Paar, A. Bauer, and M. Ulm, "Algorithmic Solution for Autonomous Vision-Based Off-Road Navigation," presented at SPIE AeroSense'98, Orlando, FL, 1998.

[9] S. Laubach, J. Burdick, and L. Matthies, "Autonomous Path-Planning for the Rocky7 Prototype Microrover," presented at International Conference on Robotics and Automation, 1998.

[10] R. Volpe, "Navigation Results from Desert Field Tests of the Rocky 7 Mars Rover Prototype," *International Journal of Robotics Research*, Submitted.

[11] Ridler and Calvard, "Picture Thresholding Using an Iterative Selection Method," *IEEE transactions on Systems, Man and Cybernetics*, 1978.

[12] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*. New York, NY: Addison-Wesley, 1992.